DDM Workload Emulator

R Vigne, E Schikuta, V Garonne, G Stewart, M Barisits, T Beermann, M Lassnig, C Serfon, L Goossens and A Nairz on behalf of the ATLAS Collaboration

1. Introduction

Data management and provisioning in the ATLAS experiment [1] is done by Don Quijote 2 (DQ2) [2] since 2006.

- 150 peta bytes
- 120 sites around the globe
- 800 active users

Although DQ2 is able to manage todays workload, it is almost at its limits:

- the amount of data has increased
- applications depending on distributed data provision (e.g. the ATLAS Production and Distributed Analysis System (PanDA) [3]) became more powerful
- various adaptions and changed requirements over the last years did compromise its basic design

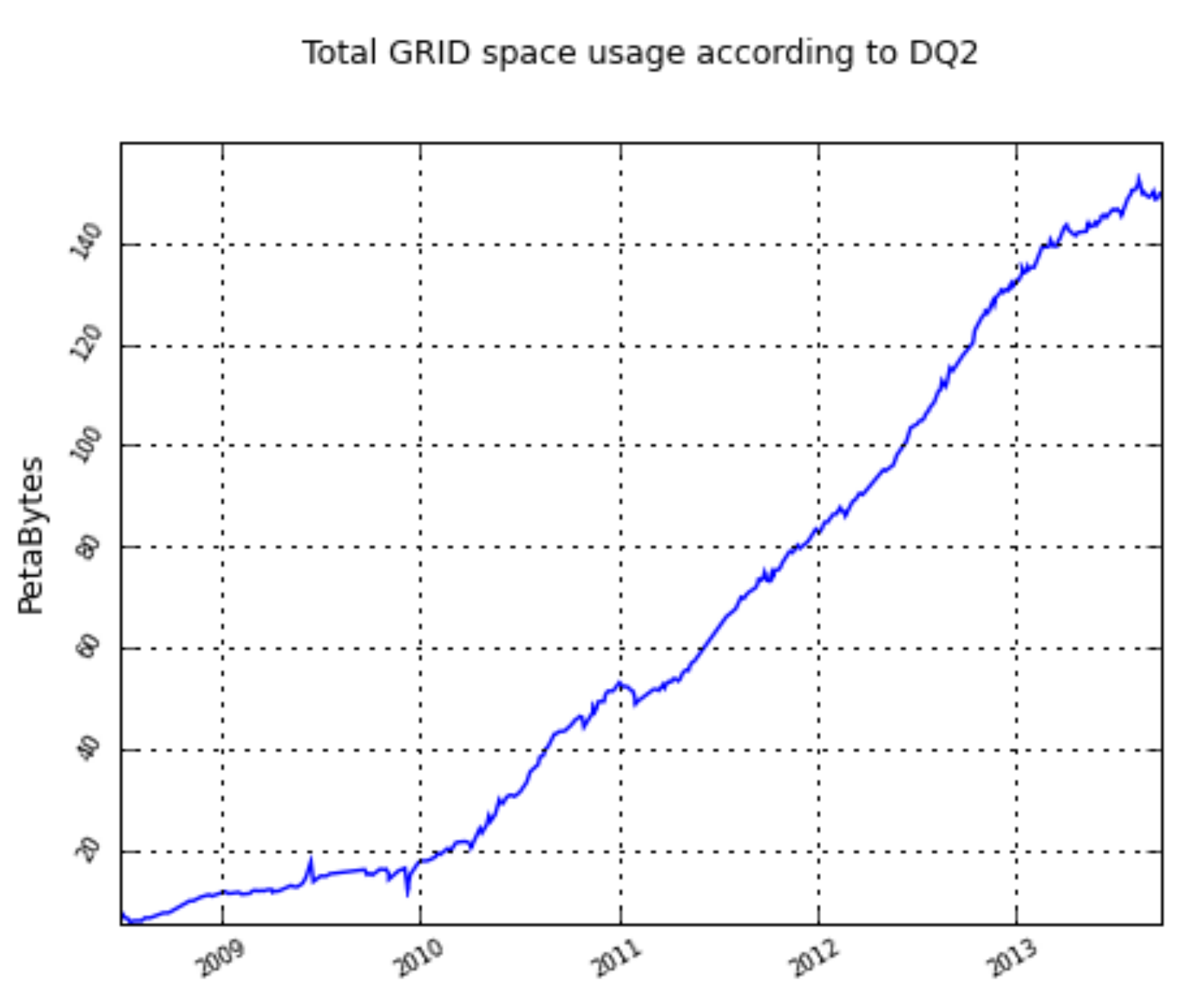


Figure 1: Data managed by DQ2

To avoid DDM becoming the bottleneck for future applications, Rucio [4] was implemented. Its design …

- … respects the experience gained over the last 7 years with DQ2 (e.g. user behaviour, application requirements, …)
- … has a strong focus on scalability

To verify its scalability, a workload emulator was developed to validate its performance at multiples of todays load.

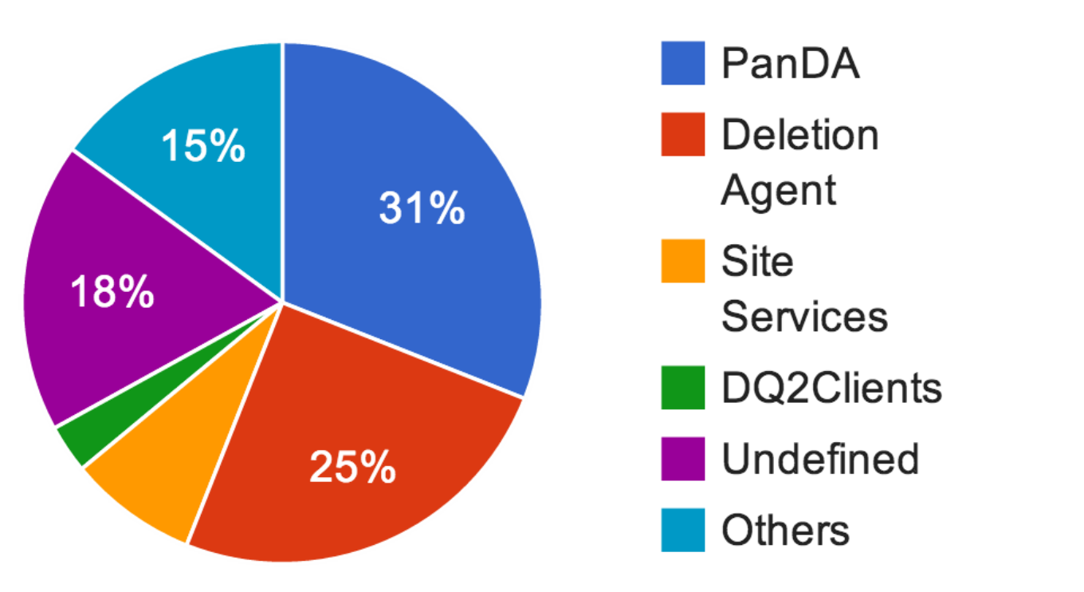
2. Profiling DQ2 Workload and Requirements

In order to gain reliable data about Rucio's scalability, we first needed to understand todays workload. We therefore analysed DQ2 log data since the beginning of 2013, namely:

- **Central File Catalogue Logs**: providing information about all API calls against DQ2 (~ 75GB/day)
- **Traces**: providing information about all file transfers performed on behalf of DQ2 (~ 25GB/day)

The output was aggregated per hour and grouped by the following attributes:

- **account**: indicating the account executing the API call
- **application ID**: an unique identifier for each application interacting with DQ2
- **method**: the called API method of DQ2



In Figure 2 we provide an overview about the top 4 applications in number of API calls. *Others* aggregate about 15 - 20 more applications, but are not mentioned explicitly.

Figure 2: Workload distribution by distinct applications.

Undefined is related to the use of outdated DQ2 client distributions, where no indication about its source or purpose is provided in the log entries.

Next we identified various use cases executed by each of these four applications. Therefore data, internal to DQ2 as well as data provided externally by the relevant applications, have been taken into account.

After the use cases have been identified, we defined "API footprints" for each of them to match against DQ2 log data, resulting in an approximation about the ratio of identified workload observed in DQ2.

Application ID	Workload Share	API Calls	Mapped to Use Cases
pandasrv	31 %	440 K	80 %
deletion-agent	25 %	360 K	100 %
site-services	8 %	110 K	100 %
dq2clients	3 %	36 K	90 %
tzero	0.07 %	1 K	100 %
Identified API Calls	**68 %**	**947 K**	**61 %**
Undefined	18 %	250 K	-
Others	15 %	200 K	-

Tabel 1: Overview of API calls mapped to use cases

3. Emulation Framework

We designed the emulation framework to put a continuous, real-world workload onto our Rucio test instance to identify performance bottlenecks and to have instant information about the performance implications of each patch set or new feature. For this purpose we identified the following requirements for the emulation framework:

- High scalability (up to multiple kHz)
- Easy extendable with new use cases (Plug-In like)
- Real-time and comprehensive monitoring of the workload and performance indicators (e.g. use case frequencies, method response times, …)

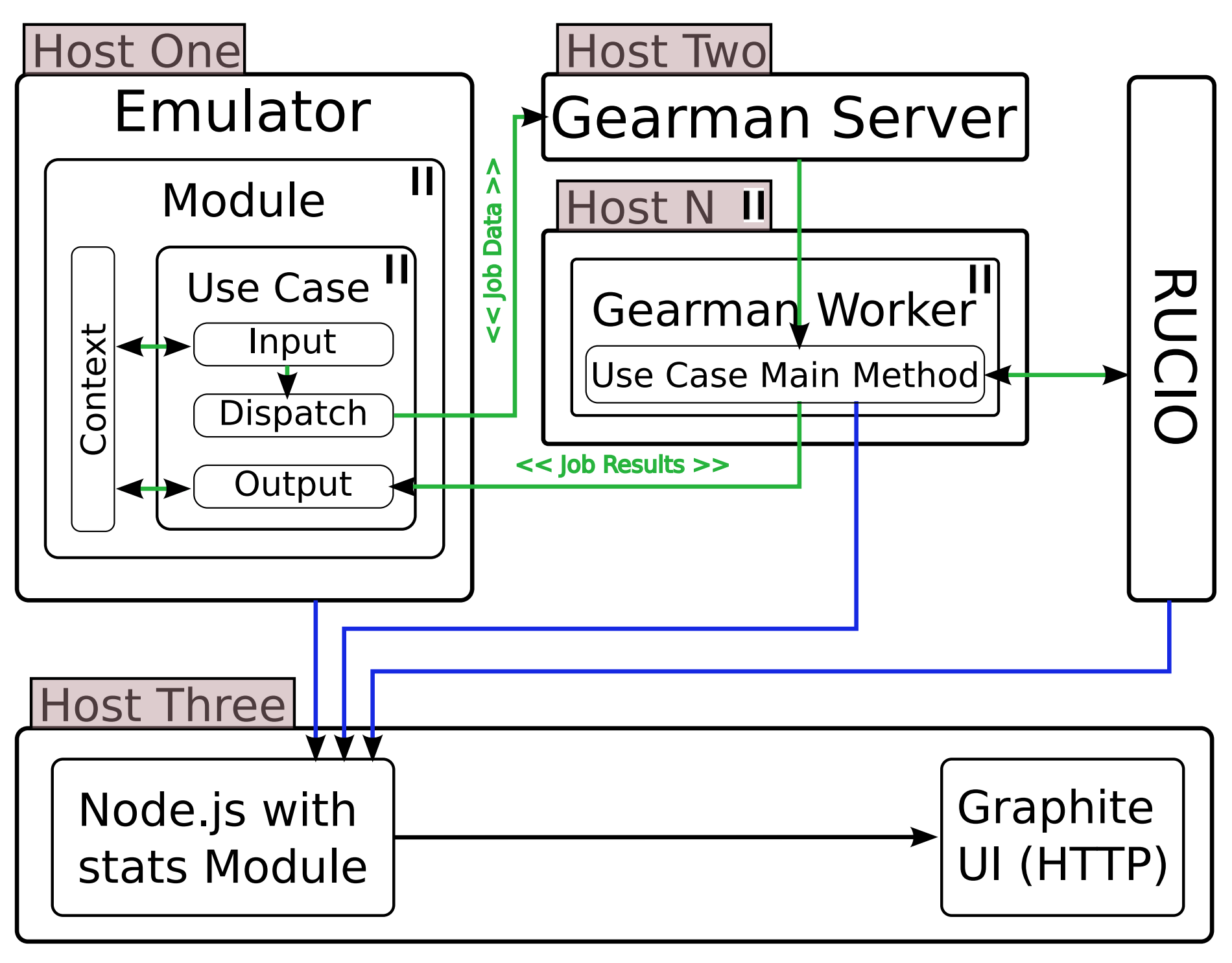


Figure 3: Overview of the emulation framework architecture

Figure 3 provides an overview about the architecture we identified to be feasible for the requirements identified above.

3.1. The Emulator

The Emulator is in charge of dispatching job descriptions (i.e. use cases) in real-time into a distributed queue (Gearman Server). It further provides …

- … a *shared context* object per module.
- … an *input* and *output* method per use case to access the context object (enabling correlated use cases).
- … a *setup* and *shutdown* method per module (e.g. loading / persisting the context object to avoid ramp-ups).
- … automatic distribution of modules over multiple processes and threads for high resource efficiency.

3.2. Gearman Framework

The Gearman framework [5] consists of Servers and Workers.

- **Gearman Server**: provides a distributed First-In-First-Out Queue for job descriptions.
- **Gearman Workers**: pick up jobs (i.e. use cases) from the server queue and report back the outcome of the execution.

As new workers can be started at any time on any host, it provides excellent horizontal scalability.

3.3. Monitoring

We decided to use Graphite [6] for data monitoring as it provides excellent capabilities for real-time data taking combined with a powerful user interface to compose various plots to illustrate the recorded data. To support data taking at frequencies higher than 1Hz, Node.js [7] is put in front, acting as an aggregator.

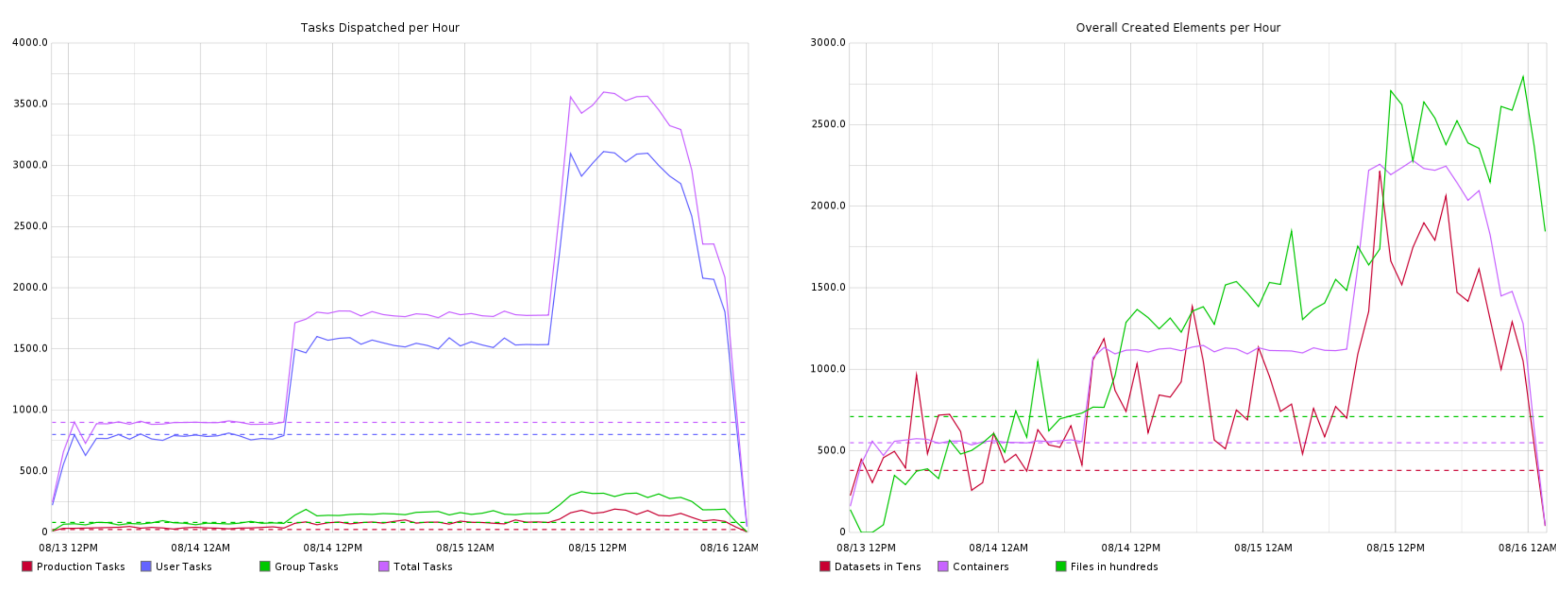


Figure 4: Scaling test with loads 1, 2, and 4

4. Conclusion

These three components (Emulator, Gearman Framwork, and Graphite+Node.js) together provide a highly scalable and powerful framework to keep a constant, and realistic workload onto Rucio. The detailed information about the run-time behaviour of the system is of great value while optimising Rucio for production.

http://rucio.cern.ch

http://atlas.ch

References

[1] CERN 2002 The ATLAS Experiment URL http://atlas.ch/

[2] Branco M, Zaluska E, de Roure D, Lassnig M and Garonne V 2010 Concurrency and Computation: Practice and Experience 22 1338–1364 URL http://dx.doi.org/10.1002/cpe.1489

[3] Maeno T, De K, Wenaus T, Nilsson P, Stewart G a, Walker R, Stradling a, Caballero J, Potekhin M and Smith D 2011 Journal of Physics: Conference Series 331 072024 ISSN 1742-6596 URL http://stacks.iop.org/1742-6596/331/i=7/a=072024?key=crossref.349ff5d6b96b79f6d1ebe3fa760f9437

[4] CERN 2013 RUCIO URL http://rucio.cern.ch

[5] Ewart J 2013 Instant Parallel Processing with Gearman (Packt Publishing Ltd) ISBN 978-1783284078

[7] Graphite - Scalable Realtime Graphing URL http://graphite.wikidot.com/

[8] Teixeira P 2012 Professional Node. js: Building Javascript Based Scalable Software (Indianapolis, Indiana, USA: John Wiley & Sons, Inc.) ISBN 978-1118185469